New Polynomial Classes for #2SAT Established Via Graph-Topological Structure

Guillermo De Ita, Pedro Bello, Meliza Contreras *

Abstract—**We address the problem of designing efficient procedures for counting models of Boolean formulas and, in this task, we establish new classes of instances where #2SAT is solved in polynomial time. Those instances are recognized by the topological structure of the underlying graph of the instances. We show that, if the depth-search over the constrained graph of a formula generates a tree where the set of fundamental cycles are disjointed (there are not common edges between any pair of fundamental cycles), then #2SAT is tractable. This class of instances do not set restrictions on the number of occurrences of a variable in a Boolean formula. Our proposal can be applied to impact directly in the reduction of the complexity time of the algorithms for other counting problems.**

Keywords:#SAT Problem, Counting models, Fibonacci Numbers.

1 Introduction

The propositional *Satisfiability* problem (SAT problem) is a special concern to the Artificial Intelligence (AI) field, and it has a direct relationship to Automated Theorem Proving. As is well known, the SAT problem is a classical NP-complete problem, and an intensive area of research has been the identification of restricted cases for which the SAT problem, as well as its optimization and counting version: MaxSAT and #SAT problems, can be solved efficiently.

The problem of counting models for a Boolean formula (#SAT problem) can be reduced to several different problems in approximate reasoning. For example, for estimating the degree of reliability in a communication network, computing degree of belief in propositional theories, for the generation of explanations to propositional queries, in Bayesian inference, in a truth maintenance systems, for repairing inconsistent databases [1, 5, 6, 14, 16]. The previous problems come from several AI applications such as planning, expert systems, approximate reasoning, etc.

For example, if we have a knowledge base KB which describes a real world W faithfully, given a formula Σ such that neither Σ nor $\overline{\Sigma}$ is a consecuence of KB, a reasonable assumption is that if more models of KB assert Σ, the more likely is that Σ will be true in W [13].

#SAT is at least as hard as the SAT problem, but in many cases, even when SAT is solved in polynomial time, no computationally efficient method is known for #SAT. For example, 2-SAT problem (SAT restricted to consider ($\leq$ 2)-CF's), it can be solved in linear time. However, the corresponding counting problem #2-SAT is a #P-complete problem.

Earlier works on #2SAT include papers by Dubois [8], Zhang [17] and Littman [9]. More recently, new upper bounds for exact deterministic algorithms for #2SAT have been found by Dahllöf [4], Fürer [11], Angelsmark [1] and Jonsson [12]. And given that #2SAT is a #P-complete problem, all the above proposals are part of the class of exponential algorithms.

The maximum polynomial class recognized for #2SAT is the class $(\leq 2, 2\mu)$-CF (conjunction of binary or unary clauses where each variable appears twice at most) [14, 15]. Here, we extend such class for considering the topological structure of the undirected graph induced by the restrictions (clauses) of the formula.

We extend the procedures presented in [6] for determining a general class of 2-CF where #2SAT is tractable. We show that a larger polynomial class for #2SAT (and for the counting independent sets too), is not restricted by the number of occurrences of the variables in the formula Σ (or the degree of its respective constrained graph G_Σ), but rather by the topological structure of G_Σ.

In a general way, if G_Σ can be expressed as a tree union a set of disjointed cycles, then $\#SAT(\Sigma)$ is computed in polynomial time. We have called to this new polynomial class "Topologically Ordered" and generalizes the polynomial classes for #2SAT presented in [6, 14, 15, 16], it allows to establish a finer border between the instances of #2SAT which are in FP or in #P. Thus, the last algorithm presented here, can be used to impact directly in the reduction of the complexity time of the algorithms for #SAT as well as for many others counting hard problems.

*Universidad Autónoma de Puebla, Faculty of Computer Sciences Av. San Claudio and 14 Sur, Puebla, México 72570 Tel: 01 (222) 2 229 55 00 ext 7239 and 7217 Email: deita@ccc.inaoep.mx, pbello@cs.buap.mx, mel_22281@hotmail.com

2 Procedure Description

2.1 Notation and Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n *Boolean variables*. A *literal* is either a variable x or a negated variable $\overline{x}$. As is usual, for each $x \in X$, $x^0 = \overline{x}$ and $x^1 = x$. We use $\upsilon(l)$ to indicate the variable involved by the literal l.

A *clause* is a disjunction of different literals (sometimes, we also consider a clause as a set of literals). For $k \in I\!N$, a *k-clause* is a clause consisting of exactly k literals and, a *$(\leq k)$-clause* is a clause with k literals at most. A unary clause has just one literal and a binary clause has exactly two literals. The empty clause signals a contradiction. A clause is *tautological* if it contains a complementary pair of literals. From now on, we will consider just non-tautological and non-contradictory clauses. A variable $x \in X$ *appears* in a clause c if either x or $\overline{x}$ is an element of c. Let $\upsilon(c) = \{x \in X : x \text{ appears in } c\}$.

A *Conjunctive Form* (CF) is a conjunction of clauses (we also consider a CF as a set of clauses). We say that Σ is a monotone CF if all of its variables appear in unnegated form. A k-CF is a CF containing only k-clauses and, $(\leq k)$-CF denotes a CF containing clauses with at most k literals. A $k\mu$-CF is a formula in which no variable occurs more than k times. A $(k, s\mu)$-CF, $(\leq k, s\mu)$-CF is a k-CF, $(\leq k) - CF$, such that each variable appears no more than s times. In this sense we have a hierarchy given by the number of occurrences by variable, where $(k, s\mu)$-CF is a restriction of $(k, (s+1)\mu)$-CF, and a hierarchy given by the number of literals by clause, where $(\leq k, s\mu)$-CF is a restriction of $(\leq (k+1), s\mu)$-CF. For any CF Σ, let $\upsilon(\Sigma) = \{x \in X : x \text{ appears in any clause of } \Sigma\}$.

An assignment s for Σ is a function $s : \upsilon(\Sigma) \to \{0, 1\}$. An *assignment* can be also considered as a set of no complementary pairs of literals. If $l \in s$, being s an assignment, then s makes l *true* and makes $\overline{l}$ *false*. A clause c is *satisfied* by s if and only if $c \cap s \neq \emptyset$, and if for all $l \in c$, $\overline{l} \in s$ then s falsifies c.

A CF F is *satisfied* by an assignment s if each clause in F is satisfied by s and s falsifies c if c is not satisfied by s and F is contradicted if it is not satisfied.

Let $SAT(\Sigma)$ be the set of models that Σ has over $\upsilon(\Sigma)$. Σ is a *contradiction* or *unsatisfiable* if $SAT(\Sigma) = \emptyset$. Let $\mu_{\upsilon(\Sigma)}(\Sigma) = |SAT(\Sigma)|$ be the cardinality of $SAT(\Sigma)$. Given Σ a CF, the SAT problem consists in determining if Σ has a model. The #SAT consists of counting the number of models of F defined over $\upsilon(\Sigma)$. We will also denote $\mu_{\upsilon(\Sigma)}(\Sigma)$ by #SAT(Σ). When $\upsilon(\Sigma)$ will clear from the context, we will omit it as a subscript.

Let $\#LANG$-SAT be the notation for the #SAT problem for propositional formulas in the class $LANG$-CF, e.g. #2-SAT denotes #SAT for formulas in 2-CF, while $\#(2, 2\mu)$-SAT denotes #SAT for formulas in the class $(2, 2\mu)$-CF. FP denotes the class of functions calculable in deterministic polynomial time, while #P is the class of functions calculable in nondeterministic polynomial time. The #SAT problem is a classical #P-complete problem.

The Graph Representation of a 2-CF

Let Σ be a 2-CF, the *constrained graph* of Σ is the undirected graph $G_\Sigma = (V, E)$, with $V = \upsilon(\Sigma)$ and $E = \{(\upsilon(x), \upsilon(y)) : (x, y) \in \Sigma\}$, that is, the vertices of G_Σ are the variables of Σ and for each clause (x, y) in Σ there is an edge $(\upsilon(x), \upsilon(y)) \in E$. The degree of a node $v \in V$ is the number of incident edges to v.

Given a 2-CF Σ, a *connected component* of G_Σ is a maximal subgraph such that for every pair of vertices x, y, there is a path in G_Σ from x to y. We say that the set of *connected components* of Σ are the subformulas corresponding to the connected components of G_Σ. We will denote $[\![n]\!] = \{1, 2, ..., n\}$.

If $\{G_1, \ldots, G_r\}$ is a partition in connected components of Σ, then:

$$\mu_{\upsilon(\Sigma)}(\Sigma) = \left[\mu_{\upsilon(G_1)}(G_1)\right] * \cdots * \left[\mu_{\upsilon(G_r)}(G_r)\right] \quad (1)$$

In order to compute $\mu(\Sigma)$, first we should determine the set of connected components of Σ, and this procedure is done in linear time [15]. Then, compute $\mu(\Sigma)$ is translated to compute $\mu_{\upsilon(G)}(G)$ for each connected component G of Σ. From now on, when we mention a formula Σ, we suppose that Σ is a connected component. We say that a 2-CF Σ is a *cycle*, a *chain* or a *tree* if G_Σ is a cycle, a chain or a tree, respectively.

Let Υ be a monotone 2-CF defined over the set of n variables $X = \{x_1, ..., x_n\}$, and let $S_I = \{x_j : j \in I\}$ be an independent set in G_Υ, that is, if no pair of vertices of S_I is joined by an edge of G_Υ, then the assignment defined by $x_i = \begin{cases} 0 & \text{if } i \in I, \\ 1 & \text{otherwise} \end{cases}$ satisfies Υ. The reason is that for every clause $(x_i \vee x_j)$ of Υ, at least one of the variables is assigned to 1. Otherwise, by the definition of Υ, $(x_i, x_j) \in E$, but both x_i and x_j are in S_I.

This shows that to compute the number of models for Σ can be alternatively reduced to compute the number of independent sets of the constrained graph G_Σ, then the remark follows easily.

Remark 1 *The problem of counting the number of models in a monotone $(2, k\mu)$-CF Σ is equivalent to counting the number of independent sets in its constrained graph G_Σ which has degree k*

2.2 Linear Procedures for $\#2SAT$

Our purpose is to identify any restriction over the class of ($\leq$ 2)-CF's under which the hard problem $\#2SAT$ becomes easy. We suppose that G_Σ is the *constrained graph* of a connected component type given by Σ a (≤ 2)-CF. We present the different typical simple graphs for G_Σ and we design linear procedures to compute $\#SAT(\Sigma)$ for those graphs.

2.2.1 If G_Σ is a Chain

First, let us consider that $G_\Sigma = (V, E)$ **is a linear chain**. Let us write down its associated formula Σ, without a loss of generality (ordering the clauses and its literals, if it were necessary), as: $\Sigma = \{c_1, ..., c_m\} = \left\{\{x_1^{\epsilon_1}, x_2^{\delta_1}\}, \{x_2^{\epsilon_2}, x_3^{\delta_2}\}, \ldots, \{x_{m-1}^{\epsilon_m}, x_m^{\delta_m}\}\right\}$, where $|\upsilon(c_i) \cap \upsilon(c_{i+1})| = 1$, $i \in [\![m-1]\!]$, and $\delta_i, \epsilon_i \in \{0, 1\}$, $i = 1, ..., m$.

As Σ has m clauses then $|\upsilon(\Sigma)| = n = m + 1$. We will compute $\mu(\Sigma)$ in base to build a series (α_i, β_i), $i = 1, ..., m$, where each pair of the series is associated to the variable x_i of $\upsilon(\Sigma)$. The value α_i indicates the number of times that the variable x_i is 'true' and β_i indicates the number of times that the variable x_i takes value 'false' over the set of models of Σ.

Let f_i be a family of clauses of Σ built as follows: $f_0 = \emptyset$,$f_i = \{c_j\}_{j \leq i}$, $i \in [\![m]\!]$. Note that $f_i \subset f_{i+1}$, $i \in [\![m-1]\!]$. Let $SAT(f_i) = \{s : s \text{ satisifies } f_i\}$, $A_i = \{s \in SAT(f_i) : x_i \in s\}$, $B_i = \{s \in SAT(f_i) : \overline{x}_i \in s\}$. Let $\alpha_i = |A_i|$; $\beta_i = |B_i|$ and $\mu_i = |SAT(f_i)| = \alpha_i + \beta_i$. From the total number of models in $\mu_i, i \in [\![m]\!]$, there are α_i of which x_i takes the logical value 'true' and β_i models where x_i takes the logical value 'false'.

For example, $c_1 = (x_1^{\epsilon_1}, x_2^{\delta_1})$, $f_1 = \{c_1\}$, and $(\alpha_1, \beta_1) = (1, 1)$ since x_1 can take one logical value 'true' and one logical value 'false' and with whichever of those values satisfies the subformula f_0 while $SAT(f_1) = \{x_1^{\epsilon_1} x_2^{\delta_1}, x_1^{1-\epsilon_1} x_2^{\delta_1}, x_1^{\epsilon_1} x_2^{1-\delta_1}\}$, and then $(\alpha_2, \beta_2) = (2, 1)$ if δ_1 were 1 or rather $(\alpha_2, \beta_2) = (1, 2)$ if δ_1 were 0.

In general, we compute the values for (α_i, β_i) associated to each node x_i, $i = 2, .., m$, according to the signs (ϵ_i, δ_i) of the literals in the clause c_i, by the next recurrence equation:

$$(\alpha_i, \beta_i) = \begin{cases} (\beta_{i-1}, \alpha_{i-1} + \beta_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (0, 0) \\ (\alpha_{i-1} + \beta_{i-1}, \beta_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (0, 1) \\ (\alpha_{i-1}, \alpha_{i-1} + \beta_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (1, 0) \\ (\alpha_{i-1} + \beta_{i-1}, \alpha_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (1, 1) \end{cases} \quad (2)$$

Note that, as $\Sigma = f_m$ then $\mu(\Sigma) = \mu_m = \alpha_m + \beta_m$. We denote with $' \rightarrow'$ the application of one of the four rules of the recurrence (2), so, the expression $(2, 3) \rightarrow (5, 2)$ denotes the application of one of the rules (in this case, the rule 4), over the pair $(\alpha_{i-1}, \beta_{i-1}) = (2, 3)$ in order to obtain $(\alpha_i, \beta_i) = (\alpha_{i-1} + \beta_{i-1}, \alpha_{i-1}) = (5, 3)$.

Example 1 *Let* $\Sigma = \{(x_1, x_2), (\overline{x}_2, \overline{x}_3), (\overline{x_3}, \overline{x}_4), (x_4, \overline{x}_5), (\overline{x}_5, x_6)\}$ *be a 2-CF which conforms a chain, the series* $(\alpha_i, \beta_i), i \in [\![6]\!]$, *is computed as:* $(\alpha_1, \beta_1) = (1, 1) \rightarrow (\alpha_2, \beta_2) = (2, 1)$ *since* $(\epsilon_1, \delta_1) = (1, 1)$, *and the rule 4 has to be applied. In general, applying the corresponding rule of the recurrence (2) according to the signs expressed by* $(\epsilon_i, \delta_i), i = 3, ..., 6$, *we have* $(2, 1) \rightarrow (\alpha_3, \beta_3) = (1, 3) \rightarrow (\alpha_4, \beta_4) = (3, 4) \rightarrow (\alpha_5, \beta_5) = (3, 7) \rightarrow (\alpha_6, \beta_6) = (10, 7)$, *and then,* $\#SAT(\Sigma) = \mu(\Sigma) = \mu_6 = \alpha_6 + \beta_6 = 10 + 7 = 17$.

If Σ is a chain, we apply (2) in order to compute $\mu(\Sigma)$. The procedure has a linear time complexity over the number of variables of Σ, since (2) is applied while we are traversing the chain, from the initial node y_0 to the final node y_m.

There are other procedures for computing #SAT(Σ) when Σ is a $(2, 2\mu)$-CF [14, 15], but such proposals do not distinguish the number of models in which a variable x takes value 1 of the number of models in which x takes value 0, situation which is made explicit in our procedure through the pair (α, β) labeled by x. This distinction over the set of models of Σ is essential when we want to extend the computing of #SAT(Σ) for more complex formulas.

Example 2 *Suppose now a* monotone 2*-CF* Υ *with m clauses and where* G_Υ *is a linear chain. I.e* $\Upsilon = \{(x_1, x_2), (x_2, x_3), \ldots, (x_{m-1}, x_m)\}$. *Then, at the beginning of the recurrence (2),* $(\alpha_1, \beta_1) = (1, 1)$ *and* $(\alpha_2, \beta_2) = (2, 1)$ *since* $(\epsilon_1, \delta_1) = (1, 1)$, *and in general, as* $(\epsilon_i, \delta_i) = (1, 1)$, *for* $i \in [\![m]\!]$, *then the rule:* $(\alpha_i, \beta_i) = (\alpha_{i-1} + \beta_{i-1}, \alpha_{i-1})$ *is always applied while we are scanning each node of the chain, thus the Fibonacci numbers appear!.*

$$\begin{array}{lll} \mu_2 & = \alpha_2 + \beta_2 & = \alpha_1 + \beta_1 + \alpha_1 = 3, \\ \mu_3 & = \alpha_3 + \beta_3 & = \mu_2 + \alpha_2 = 5, \\ (\mu_i)_{i \geq 3} & = \alpha_i + \beta_i & = \mu_{i-1} + \mu_{i-2} \end{array}$$

Applying the Fibonacci series for the formula F of the example 2, we obtain the values $(\alpha_i, \beta_i), i = 1, ..., 6$: $(1, 1) \rightarrow (2, 1) \rightarrow (3, 2) \rightarrow (5, 3) \rightarrow (8, 5) \rightarrow (13, 8)$, and this last series coincides with the Fibonacci numbers: $(F_2, F_1) \rightarrow (F_3, F_2) \rightarrow (F_4, F_3) \rightarrow (F_5, F_4) \rightarrow (F_6, F_5) \rightarrow (F_7, F_6)$. We infer that $(\alpha_i, \beta_i) = (F_{i+2}, F_{i+1})$ and then $\mu_i = F_{i+2} + F_{i+1} = F_{i+3}, i = 1, ..., m$. E.g. for $m = 5$, we have $\mu(F) = \mu_5 = F_7 + F_6 = F_8 = 21$.

Theorem 1 *Let* Σ *be a monotone 2-CF with m clauses such that* G_Σ *is a chain, then:*

$$\#SAT(\Sigma) = F_{m+3}$$

Corollary 1 *If G_F is a chain with m edges then the number of independent sets in G_F is F_{m+3}.*

Considering Parallel Edges

Consider the case where in a Conjunctive Form there are two 2-clauses involving the same variables. In this case, the constrained graph has two parallel edges and the recurrence equation in (2) has to consider four different signs.

Suppose, for example that we have the two clauses: $c_k = (x_{i-1}^{\epsilon_k}, x_i^{\delta_k})$ and $c_j = (x_{i-1}^{\epsilon_j}, x_i^{\delta_j})$ which involve the two variables: x_{i-1} and x_i. Then, we compute the values for (α_i, β_i) associated to the node x_i, according to the signs (ϵ_k, δ_k) and (ϵ_j, δ_j) associated to the clauses: c_k and c_j in the following way:

(α_i, β_i) takes the following six possible values:

$(\alpha_{i-1}, \alpha_{i-1})$ if $(\epsilon_k, \delta_k) = (1,1)$ *and* $(\epsilon_j, \delta_j) = (1,0)$
$(\mu_{i-1}, 0)$ if $(\epsilon_k, \delta_k) = (1,1)$ *and* $(\epsilon_j, \delta_j) = (0,1)$
$(\beta_{i-1}, \alpha_{i-1})$ if $(\epsilon_k, \delta_k) = (1,1)$ *and* $(\epsilon_j, \delta_j) = (0,0)$
$(\alpha_{i-1}, \beta_{i-1})$ if $(\epsilon_k, \delta_k) = (1,0)$ *and* $(\epsilon_j, \delta_j) = (0,1)$
$(0, \mu_{i-1})$ if $(\epsilon_k, \delta_k) = (1,0)$ *and* $(\epsilon_j, \delta_j) = (0,0)$
$(\beta_{i-1}, \beta_{i-1})$ if $(\epsilon_k, \delta_k) = (0,1)$ *and* $(\epsilon_j, \delta_j) = (0,0)$

And Considering the case when in a Conjunctive Form there are three 2-clauses involving the same variables. In such case, the constrained graph has three parallel edges. Suppose, for example that we have the following three clauses: (x_{i-1}, x_i),$(x_{i-1}, \overline{x}_i)$,$(\overline{x}_{i-1}, \overline{x}_i)$ then $(\alpha_i, \beta_i) = (0, \alpha_{i-1})$ since $(\overline{x}_{i-1}, x_i)$ is the unique clause which was not considered. And then, only the true values for x_{i-1} are preserved and used for assigning the false values to x_i.

This means, that the negation of the clause which does not appear indicates the value for (α_i, β_i). For example, $\overline{(\overline{x}_{i-1}, x_i)} = x_{i-1} \wedge \overline{x}_i$ and then only the true values for x_{i-1} have to be changed to false values for x_i in order to satisfy the previous three clauses, simultaneously. And this pattern is used for the other four forms to choose three clauses with the same two variables.

Processing Unary Clauses

If there are unary clauses in Σ, i.e. $U \subseteq \Sigma$ and $U = \{(l_1), (l_2), ..., (l_k)\}$. Then, when the recurrence (2) is being applied over a node x_i of G_Σ, it has to be checked if $x_i \in \upsilon(U)$ or not. If $x_i \notin \upsilon(U)$ we only apply the recurrence (2), but if $x_i \in \upsilon(U)$ then

$$(\alpha_i, \beta_i) = \begin{cases} (0, \beta_i) & \text{if} \quad (\overline{x}_i) \in U \\ (\alpha_i, 0) & \text{if} \quad (x_i) \in U \\ (0,\ 0) & \text{if} \quad (x_i) \in U \wedge (\overline{x}_i) \in U \end{cases}$$

Since an unary clause uniquely determines the values of its variable. Furthermore when $(x_i) \in U$ and $(\overline{x}_i) \in U$ then the original formula is insatisfactible and then $\mu(\Sigma \cup U)$=0.

Of course, the previous cases: parallel edges and unary clauses can be considered in a pre-processing of the formula before applying the general algorithm.

2.2.2 If G_Σ is a Tree

Let Σ be a Boolean formula with n variables and m clauses and where there are no cycles in $G_\Sigma = (V, E)$. Traversing G_Σ in depth-first build a tree, that we denote as A_Σ, whose root node is any vertex $v \in V$, e.g. the node with minimum degree in G_Σ, and where v is used for beginning the depth-first search.

We denote with (α_v, β_v) the associated pair to a node v ($v \in A_\Sigma$). We compute $\mu(\Sigma)$ while we are traversing G_Σ in depth-first search, for the next procedure.

Algorithm Count_Models_for_Trees(A_Σ)

Input: A_Σ the tree defined by the depth-search over G_Σ

Output: The number of models of Σ

Procedure: Traversing A_Σ in depth-first, and when a node $v \in A_\Sigma$ is left (all of its edges have been processed), assign:

1. $(\alpha_v, \beta_v) = (1,1)$ if v is a leaf node in A_Σ.
2. If v is a father node with a list of child nodes associated, i.e., $u_1, u_2, ..., u_k$ are the child nodes of v, then as we have already visited all the child nodes, then each pair $(\alpha_{u_j}, \beta_{u_j})$ $j = 1, ..., k$ has been defined based on (2). $(\alpha_{v_i}, \beta_{v_i})$ is obtained by apply (2) over $(\alpha_{i-1}, \beta_{i-1}) = (\alpha_{u_j}, \beta_{u_j})$. This step is iterated until computes all the values $(\alpha_{v_j}, \beta_{v_j})$, $j = 1, ..., k$. And finally, let $\alpha_v = \prod_{j=1}^{k} \alpha_{v_j}$ and $\beta_v = \prod_{j=1}^{k} \beta_{v_j}$.
3. If v is the root node of A_Σ then returns$(\alpha_v + \beta_v)$.

This procedure returns the number of models for Σ in time $O(n+m)$ which is the necessary time for traversing G_Σ in depth-first.

Example 3 *Let $\Sigma = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_5), (x_4, x_6), (x_6, x_7), (x_6, x_8)\}$ be a 2-CF and consider us that the depth-search starts in the node x_1. The tree generated by the depth-search as well as the number of models in each level of the tree is showed in Figure 1. The procedure Count_Models_for_Trees returns for $\alpha_{x_1} = 41$, $\beta_{x_1} = 36$ and the total number of models is: $\#SAT(\Sigma) = 41 + 36 = 77$.*

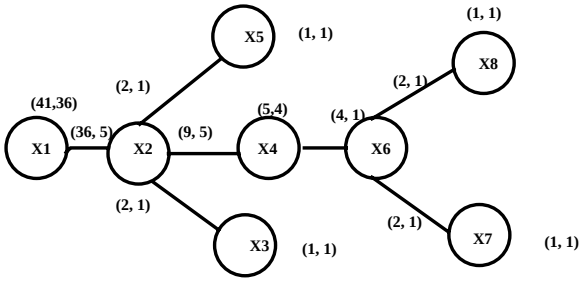


Figure 1: The tree graph for the formula of the example 3

2.2.3 If G_Σ is a Cycle

Let G_Σ be a simple cycle with m nodes, that is, all the variables in $\upsilon(\Sigma)$ appear twice, $|V| = m = n = |E|$. Ordering the clauses in Σ in such a way that $\mid \upsilon(c_i) \cap \upsilon(c_{i+1}) \mid = 1$, and $c_{i_1} = c_{i_2}$ whenever $i_1 \equiv i_2 \bmod m$, hence $x_1 = x_m$, then $\Sigma = \left\{c_i = \{x_{i-1}^{\epsilon_i}, x_i^{\delta_i}\}\right\}_{i=1}^{m}$, where $\delta_i, \epsilon_i \in \{0, 1\}$. Decomposing Σ as $\Sigma = \Sigma' \cup c_m$, where $\Sigma' = \{c_1, ..., c_{m-1}\}$ is a chain and $c_m = (x_{m-1}^{\epsilon_m}, x_1^{\delta_m})$ is the edge which conforms with $G_{\Sigma'}$ the simple cycle: $x_1, x_2, ..., x_{m-1}, x_1$. We can apply the linear procedure described in (2.2.1) for computing $\mu(\Sigma')$.

Every model of Σ' had determined logical values for the variables: x_{m-1} and x_1 since those variables appear in $\upsilon(\Sigma')$. Any model s of Σ' satisfies c_m if and only if ($x_{m-1}^{1-\epsilon_m} \notin s$ and $x_m^{1-\delta_m} \notin s$), that is, $SAT(\Sigma' \cup c_m) \subseteq SAT(\Sigma')$, and $SAT(\Sigma' \cup c_m) = SAT(\Sigma') - \{s \in SAT(\Sigma') : s \text{ falsifies } c_m\}$. Let $X = \Sigma' \cup \{(x_{m-1}^{1-\epsilon_m}) \wedge (x_m^{1-\delta_m})\}$, $\mu(X)$ is computed as a chain with two unary clauses, then:

$$\begin{aligned} \#SAT(\Sigma) &= \mu(\Sigma) = \mu(\Sigma') - \mu(X) \\ &= \mu(\Sigma') - \mu(\Sigma' \wedge (x_{m-1}^{1-\epsilon_m}) \wedge (x_m^{1-\delta_m})) \end{aligned} \quad (3)$$

For example, let us consider Σ be a monotone 2-CF with m clauses such that G_Σ is a simple cycle. $\Sigma = \left\{c_i = \{x_{i-1}^{\epsilon_i}, x_i^{\delta_i}\}\right\}_{i=1}^{m}$, where $\delta_i = \epsilon_i = 1$, $\upsilon(c_i) \cap \upsilon(c_{i+1}) = \{x_i\}$, and $c_{i_1} = c_{i_2}$ whenever $i_1 \equiv i_2 \bmod m$, hence $x_1 = x_m$. Let $\Sigma' = \{c_1, ..., c_{m-1}\}$, then: $\mu(\Sigma') = \mu_{m-1} = F_{m-1+3} = F_{m+2}$ for theorem 1. As $\epsilon_m = \delta_m = 1$ then $\mu(X) = \mu(\Sigma' \wedge (\overline{x}_1) \wedge (\overline{x}_{m-1}))$ which is computed by the series: $(\alpha_1, \beta_1) = (0, 1) = (F_0, F_1)$ since $(\overline{x}_1) \in X$, $(\alpha_2, \beta_2) = (1, 0) = (F_1, F_0)$; $(\alpha_3, \beta_3) = (1, 1) = (F_2, F_1)$; $(\alpha_4, \beta_4) = (2, 1) = (F_3, F_2)$; and in general $(\alpha_i, \beta_i) = (F_i, F_{i-1})$, then for the variable x_{m-1}, $(\alpha_{m-1}, \beta_{m-1}) = (F_{m-1}, F_{m-2})$, then $\mu(X) = \beta_{m-1} = F_{m-2}$ since $(\overline{x}_m) \in X$. Finally, $\#SAT(\Sigma) = \mu(\Sigma) = \mu(\Sigma') - \mu(X) = F_{m+2} - F_{m-2}$. On the other hand, $F_{m+2} - F_{m-2} = F_{m+1} + F_m - F_{m-2} = F_{m+1} + F_{m-1} + F_{m-2} - F_{m-2} = F_{m+1} + F_{m-1}$.

Theorem 2 *Let Σ be a monotone 2-CF with m clauses and where G_Σ is a simple cycle, then: $\#SAT(\Sigma) = F_{m+2} - F_{m-2} = F_{m+1} + F_{m-1}$.*

Corollary 2 *If G_Σ is a simple cycle with m nodes then the number of independent sets of G_Σ is $F_{m+1} + F_{m-1}$.*

Example 4 *Let $\Sigma = \{c_i\}_{i=1}^{6} = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_4\}, \{x_4, x_5\}, \{x_5, x_6\}, \{x_6, x_1\}\}$ be a monotone 2-CF which represents the cycle G_Σ=(V,E), see Figure 2. Let $G' = (V, E')$ where $E = E' \cup \{c_6\}$, that is, the new graph G' is Σ minus the edge c_6. Applying theorem 2, $\#SAT(\Sigma) = F_7 + F_5 = 13 + 5 = 18$*

Union of Cycles and Chains

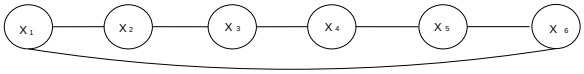
Now, Let Σ be a monotone 2-CF such that G_Σ is the union of a chain graph G_L and an edge: $\{x_j, x_k\}$ which is adyacent to two nodes of G_L, e.g. see Figure 3.

Let $L = \{(x_i, x_{i+1})\}, i = 1, ..., m$ be the formula chain and let $Y = \{(\overline{x}_j), (\overline{x}_k)\} \cup \{(x_i, x_{i+1})\}, i = 1, ..., k$. We note that $\mu(\Sigma) = \mu(L) - \mu(Y)$ since we must eliminate from the set of models of L those models that falsify the clause (x_j, x_k) in order to satisfy Σ.

We denote with $(\alpha_i, \beta_i)_{/G}$ the corresponding pair (α_i, β_i) associated to the node x_i from the constrained graph of the formula G.

In previous section, we have seen how to compute $\mu(Y)$ when Y is a chain union two unary clauses. Applying recurrence (2), in the step (j)-th, we have that $(\alpha_j, \beta_j)_{/Y} = (F_{j+2}, F_{j+1})$, and as x_j appears in nonpositive way in Y then $(\alpha_j, \beta_j)_{/Y} = (0, F_{j+1})$.

For the next steps, we have that: $(\alpha_{j+1}, \beta_{j+1})_{/Y} = (F_{j+1}, 0) \rightarrow (\alpha_{j+2}, \beta_{j+2})_{/Y} = (F_{j+1}, F_{j+1})$; $(\alpha_{j+3}, \beta_{j+3})_{/Y} = (2 \cdot F_{j+1}, F_{j+1}) \rightarrow (3 \cdot F_{j+1}, 2 \cdot F_{j+1}) = (F_4 \cdot F_{j+1}, F_3 \cdot F_{j+1})$. And for the step k: $(\alpha_k, \beta_k)_{/Y} = (F_l \cdot F_{j+1}, F_{l-1} \cdot F_{j+1})$ where l is the length of the cycle, and as x_k appears in nonpositive way in Y, then $F_l \cdot F_{j+1}$ will be changed to zero, resulting in this way that $(\alpha_k, \beta_k)_{/Y} = (0, \beta_k) = (0, F_{l-1} \cdot F_{j+1})$. Therefore $\mu(Y) = F_{l-1} \cdot F_{j+1} = F_{k-j-1} \cdot F_{j+1}$.



$$(\alpha_1,\beta_1) \rightarrow (\alpha_2,\beta_2) \rightarrow (\alpha_3,\beta_3) \rightarrow (\alpha_4,\beta_4) \rightarrow (\alpha_5,\beta_5) \rightarrow (\alpha_6,\beta_6)$$
$$(1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \rightarrow (8,5) \rightarrow (13,8)$$
$$(0,1) \rightarrow (1,0) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3)$$
$$\Rightarrow (13,8)-(0,3)=(13,5)$$

Figure 2: Computing $\#SAT(\Sigma)$ when G_Σ is a cycle

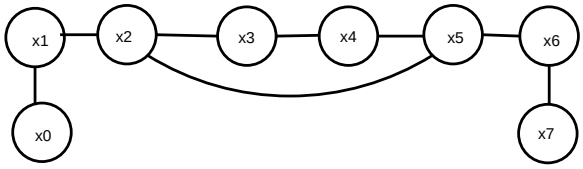


Figure 3: Constrained Graph of the formula of the example 5

While the corresponding pair (α_k,β_k) in the series $(\alpha_i,\beta_i)_{/L}$, is: $(\alpha_k,\beta_k)_{/L} = (F_{k+2},F_{k+1})$, and in order to compute $\mu(\Sigma)$, we have in the step k: $(\alpha_k,\beta_k)_\Sigma = (\alpha_k,\beta_k)_{/L} - (\alpha_k,\beta_k)_{/Y} = (F_{k+2},F_{k+1}) - (0,F_{l-1}\cdot F_{j+1}) = (F_{k+2},F_{k+1}-F_{l-1}\cdot F_{j+1}) = (F_{k+2},F_{j+l+1}-F_{l-1}\cdot F_{j+1}) = (F_{k+2},F_l\cdot F_{j+2})$.

For the following pairs of the series, we have $(\alpha_{k+1},\beta_{k+1})_{/\Sigma} = (F_{k+2}+F_l\cdot F_{j+2},F_{k+2}) \rightarrow (2\cdot F_{k+2}+F_l\cdot F_{j+2},F_{k+2}+F_l\cdot F_{j+2})$; $(\alpha_{k+3},\beta_{k+3})_{/\Sigma} = (3\cdot F_{k+2}+2\cdot F_l\cdot F_{j+2},2\cdot F_{k+2}+F_l\cdot F_{j+2})$, and in general, $(\alpha_{k+i},\beta_{k+i})_{/\Sigma} = (F_{i+1}\cdot F_{k+2}+F_i\cdot F_l\cdot F_{j+2},F_i\cdot F_{k+2}+F_{i-1}\cdot F_l\cdot F_{j+2})$.

If we consider, $m = k+p$, then: $(\alpha_m,\beta_m)_{/\Sigma} = (\alpha_{k+p},\beta_{k+p})_{/\Sigma} = (F_{p+1}\cdot F_{k+2}+F_p\cdot F_l\cdot F_{j+2},F_p\cdot F_{k+2}+F_{p+1}\cdot F_l\cdot F_{j+2})$, and therefore $\mu(\Sigma) = \alpha_m+\beta_m = F_{p+1}\cdot F_{k+2}+F_p\cdot F_l\cdot F_{j+2}+F_p\cdot F_{k+2}+F_{p+1}\cdot F_l\cdot F_{j+2} = F_{k+2}\cdot(F_{p+1}+F_p)+F_l\cdot F_{j+2}\cdot(F_p+F_{p-1})$. Thus, #SAT($\Sigma$) can be expressed as: $F_{p+2}\cdot F_{k+2}+F_{p+1}\cdot F_l\cdot F_{j+2}$, being j the start node of the cycle, k the final node of the cycle, l the length of the cycle and $p = m-k$ the number of nodes between the final node of the cycle and the final node of the chain.

Theorem 3 *If G_Σ can be decomposed in a linear chain with m edges and a simple cycle of length l between the nodes j and k, $0<j<k<m$, where there are p nodes between the node k and the node $m+1$, then the number of independent sets in G_Σ, which corresponds with the number of models of the 2-CF monotone formula Σ, is: $F_{k+2}\cdot F_{p+2}+F_l\cdot F_{j+2}\cdot F_{p+1}$*

Example 5 *Let $\Sigma = \{(x_i,x_{i+1})\}i=0,\ldots,6\cup\{(x_2,x_5)\}$. G_Σ is shown in Figure 2. G_Σ contains the simple cycle: x_2,x_3,x_4,x_5,x_2, then $j=2,k=5$ and $l=k-j+1=4$ is the length of the cycle. The linear chain embedded in G_Σ has $m=7$ edges (and the total graph has 8 edges), then $p=m-k=7-5=2$ and, according to theorem (3): $\mu(\Sigma)=F_7\cdot F_4+F_3\cdot F_4\cdot F_3=13\cdot 3+2\cdot 3\cdot 2=51$.*

2.2.4 G_Σ has Only Non-intersected Cycles

Finally, we arrive to the end of the category of connected components G_Σ where $\#SAT(\Sigma)$ can be computed in polynomial time. Let $G_\Sigma=(V,E)$ be the constrained graph of a Boolean formula Σ with n variables and m clauses. Choose arbitrarily a node $v_r\in V$ for starting the depth-first search over G_Σ in order to build the tree A_Σ and where v_r will be the root node.

Let T be a tree graph, any edge adyacent to two nodes of T conforms a simple cycle, such cyle is called a fundamental cycle and this edge is called a *back edge*. Each *back edge* that we find during the depth-first search is added to the base cycle matrix CM.

Since each back edge signals the beginning and the end of a fundamental cycle in A_Σ, we store each fundamental cycle in each row of CM, beginning from the back edge and continuing with the tree edges of the cycle, each edge in each column until we arrive to the edge which closes the base cycle.

We call *back-edge_clause* to the clause in Σ corresponding to the back edge in G_Σ. Let $C=\{c_1,\ldots,c_k\}$ be the set of fundamental cycles found during the depth first search and let $T_\Sigma = A_\Sigma\cup C$ be the depth first graph.

Given two distinct fundamental cycles C_i and C_j from the depth-first graph, if C_i and C_j share common edges we say that both cycles are *intersected*, that is, $C_i\oplus C_j$ conforms a new cycle, where $\oplus$ denotes the operation or-exclusive between the set of edges of the cycles. If the cycles C_i and C_j have no common nodes nor edges then we say that both cycles are *independent*, that is, $C_i\oplus C_j = C_i\cup C_j$. While if the cycles have not common edges but maybe they have a common node, we say that both cycles are *non-intersected*. Note that a pair of independent cycles are non-intersected.

We establish in the following theorem the larger class of formulas where the #2SAT problem can be computed in polynomial time.

Theorem 4 *Applying a depth-first search over G_Σ, if the set of fundamental cycles obtained during the search are non-intersected, then $\#SAT(\Sigma)$ is computed in polynomial time.*

We prove this theorem given the polynomial time algorithm which computes #2SAT for this class of formulas.

Algorithm Count_Models_for_Disjoint_cycles()

Input: T_Σ the tree defined by the depth-first search over G_Σ and CM the base cycle matrix.

Output: The number of models of Σ.

Procedure:

1. Translate A_Σ in a DAG (Directed Acyclic Graph), giving an orientation to each edge $\{u, v\}$ by directing $u \rightarrow v$ if v is an ancestor node of u in A_Σ. Let D_Σ be the DAG built in this way.

2. Apply a topological order procedure over D_Σ obtaining an arrangement of the vertices.

3. Computing the number of models of Σ, traversing by each node in D_Σ according to the order given by the topological procedure and computing the pair (α_v, β_v) when the node v is visited, in the following way:

a) $cycle = false$ /* Beginning without cycle */

b) Let $v = First\_Topological\_Order()$
/*v is the first node of the arrangement */

do {

c) If (v is a leaf node in T_Σ) then $(\alpha_v, \beta_v) = (1, 1)$

d) If (v has child nodes) then

e) for each child node u of v
{
Let $(\alpha_{i-1}, \beta_{i-1}) = (\alpha_u, \beta_u)$.
Apply recurrence (2), computing: $(\alpha_{i-1}, \beta_{i-1}) \rightarrow (\alpha_i, \beta_i) = (\alpha_v, \beta_v)$.
If $((\alpha_{v-old}, \beta_{v-old})$ had associated a value) then $(\alpha_v, \beta_v) = (\alpha_v * \alpha_{v-old}, \beta_v * \beta_{v-old})$
Let $(\alpha_{v-ci}, \beta_{v-ci}) = (\alpha_{v-old}, \beta_{v-old})$
/* Update values for cycles */
Let $(\alpha_{v-old}, \beta_{v-old}) = (\alpha_v, \beta_v)$
/* Keep the old values */
}

f) If (cycle) then
{
$(\alpha_{i-1}, \beta_{i-1}) = (\alpha_{u_c}, \beta_{u_c})$
/* Apply (2) also to the cycle */
$(\alpha_{i-1}, \beta_{i-1}) \rightarrow (\alpha_i, \beta_i) = (\alpha_{v_c}, \beta_{v_c})$
If($(\alpha_{v-ci}, \beta_{v-ci})$ had associated a value) then
$(\alpha_{v_c}, \beta_{v_c}) = (\alpha_{v_c} * \alpha_{v-ci}, \beta_{v_c} * \beta_{v-ci})$
}

g) If (v has two output edges)then
$cycle = true$
/*Marks the beginning of a cycle*/
If($\overline{v} \in back\_edge\_clause$) then
$(\alpha_{v_c}, \beta_{v_c}) = (1, 0)$ otherwise $(\alpha_{v_c}, \beta_{v_c}) = (0, 1)$.
/* end of a base cycle */

h) If (v closes a cycle of CM) then
{ If($\overline{v} \in back\_edge\_clause$) then
$(\alpha_{v_c}, \beta_{v_c}) = (\alpha_{v_c}, 0)$ otherwise $(\alpha_{v_c}, \beta_{v_c}) = (0, \beta_{v_c})$.
$(\alpha_{v-old}, \beta_{v-old}) = (\alpha_v, \beta_v) = (\alpha_v - \alpha_{v_c}, \beta_v - \beta_{v_c})$
$cycle = false$
}
/*Finish a cycle*/

i) Let $v = Next\_Topological\_Order(v)$;

j) }while($v <> Nil$);

k) return$(\alpha_{v_r} + \beta_{v_r})$

The procedure: $First\_Topological\_Order()$ returns the first vertice and $Next\_Topological\_Order(v)$ returns the next vertice of its argument, according to the arrangement built for the topological order procedure in step (2).

Note that all the procedures involved in $Count\_Models\_for\_Disjoint\_cycles$ such as; depth-first search, detecting if the set of fundamental cycles are disjointed, applying topological order, traversing by D_Σ according to the order given by the topological order, etc... All of them are linear procedures over the length of the graph, thus, the main procedure has complexity time $O(n + m)$ being n the number of variables and m the number of clauses of Σ. Then, our proposal is a linear time procedure over the length of the formula Σ.

In the following example, the term "cycle" is used to indicate a path in the DAG which comes from an original fundamental cycle of the matrix CM.

Example 6 *Let* $\Sigma = \{(\overline{x}_1, x_2), (x_2, x_3), (x_1, \overline{x}_{13}), (x_2, x_4), (\overline{x}_{12}, x_2), (\overline{x}_3, \overline{x}_4), (x_4, x_5), (x_4, x_{11}), (x_5, \overline{x}_6), (x_{11}, x_5), (\overline{x}_6, \overline{x}_7), (x_7, x_8), (x_7, \overline{x}_9), (x_{10}, \overline{x}_7), (x_9, x_5), (x_{13}, \overline{x}_{12})\}$.

Applying the above algorithm over Σ and considering to x_1 as the root node of the tree A_Σ. The step (1) builds the DAG D_Σ showed in Figure 4. The order of evaluation of

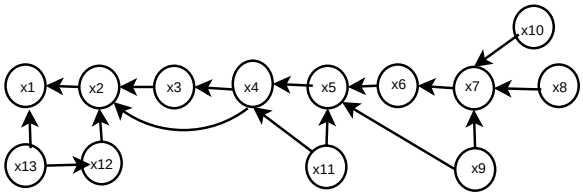


Figure 4: The DAG built in base of the formula of the example 5

the nodes of the DAG according to the Topological Order*, step (2), is performed in step (3), in the following way:*

Evaluate leaf nodes (step c)*:* x_{10}, x_8, x_9*. The node has two output edges* (step g)*:* x_9*. Evaluate father with many children* (step e)*:* $x_{10}; x_8; x_9 \rightarrow x_7$*. Evaluate path and cycle* (steps e and f)*:* x_9, x_7, x_6, x_5*. Evaluate end of a cycle* (step h)*:* x_5*. The node has two output edges* (step g)*:* x_{11}*. Evaluate path and cycle* (steps e and f)*:* x_{11}, x_5, x_4*. Evaluate end of a cycle* (step h)*:* x_4*. Evaluate path and cycle* (steps e and f)*:* x_4, x_3, x_2*. Evaluate end of a cycle* (step h)*:* x_2*. Evaluate path and cycle* (steps e and f)*:* x_{13}, x_{12}, x_2, x_1*. Evaluate end of a cycle* (step h)*:* x_1*.*

Thus, the procedures presented here, for computing $\mu(\Sigma)$ being Σ a Boolean formula in 2-CF and where G_Σ is a cycle, a chain, a tree or a tree union non-intersected cycles, each one runs in linear time over the length of the given formula.

We have called to the class of formulas which holds the conditions of theorem (4) a *topologically ordered* formulas. This class of formulas is a superclass of $(2, 2\mu)$-CF, and it has not restriction over the number of occurrences of a variable over the formulas, although $(2, 3\mu)$-CF is a $\#P$-complete problem.

The class of *topologically ordered* Boolean formulae brings us a new paradigm for solving #SAT in polynomial time, and it would impact directly on the complexity time of the algorithms for #SAT.

The dichotomy theorem [3] establishes that if F is a finite set of affine logical relations then #SAT(F) is in FP, otherwise #SAT(F) is #P-complete. This dichotomy theorem fails to show that the #2SAT problem can be solved in polynomial time for the topologically ordered formulas. In fact, this new class of topologically ordered formulas demands for a deep study of the topological structure of the Boolean formulae (or of the associated graphs) which allows to obtain a finer border between the two classes: FP and #P, for the #2SAT problem.

On the other hand, we also determine a new polynomial class of graphs for counting the number of independent sets, such is the case where the considered 2-CF is monotone and holds the condition of theorem 5.

Theorem 5 *Given an undirected connected graph* $G = (V, E)$*, if it is acyclic or has a set of disjointed cycles, then the number of independent sets is computed in polynomial time.*

3 Conclusions and Future Work

#SAT for the class of Boolean formulas in 2-CF is a classical #P-complete problem. Until now, the maximum subclass of 2-CF where #2SAT is solved efficiently is for the class $(2, 2\mu)$-CF, which are the Boolean formulas in 2-CF where each variable appears twice at most.

We present different linear procedures to compute #SAT for subclasses of 2-CF. We have called *topologically ordered* to the formulas Σ in 2-CF where G_Σ (the constrained undirected graph of Σ) is acyclic or a tree union non-intersected cycles. And we have shown that $\#SAT(\Sigma)$ is computed in linear time over the length of the formula Σ for the class of *topologically ordered* formulas.

The latter class of 2-CF contains the class $(2, 2\mu)$-CF, and it does not have restrictions over the number of occurrences of a variable in the given formula, although $(2, 3\mu)$-SAT is a $\#P$-complete problem. Then, the class of *topologically ordered* formulas brings us a new paradigm for solving #SAT, and would be used to incide directly over the complexity time of the algorithms for #2SAT.

Furthermore, we have shown a strong relation between the number of models of a monotone formula in 2-CF and the Fibonacci numbers which allows to determine new classes of graphs where the number of independent sets is computed in polynomial time.

References

[1] Angelsmark O., Jonsson P., Improved Algorithms for Counting Solutions in Constraint Satisfaction Problems, *In ICCP: Int. Conf. on Constraint Programming*, 2003.

[2] Bulatov, A. , Dalmau V., *Towards a dichotomy theorem for the counting constraint satisfaction problem*, Technical report, Computing Lab., Oxford Univ., 2003. Available from web.comlab.ox.ac.uk/oucl/work/andrei.bulatov/counting.ps

[3] Creignou N., Hermann M., "Complexity of Generalized Satisfiability Counting Problems", *Information and Computation* pp. 1-12, N125, 1996

[4] Dahllöf, V., Jonsonn, P., Wahlström, M., "Counting models for 2SAT and 3SAT formulae.", *Theoretical Computer Sciences 332*, pp. 265-291, N332(1-3), 2005.

[5] Darwiche, A., "On the Tractability of Counting Theory Models and its Application to Belief Revision and Truth Maintenance", *Jour. of Applied Nonclassical Logics*, pp. 11-34, N11(1-2),2001.

[6] De Ita G., "Polynomial Classes of Boolean Formulas for Computing the Degree of Belief", *Advances in Artificial Intelligence LNAI 3315*, pp.430-440,2004.

[7] Dyer, M., Greenhill C., *Some #P-completeness Proofs for Coulorings and Independent Sets*, Research Report Series, University of Leeds, 1997.

[8] Dubois, O., " Counting the number of solutions for instances of satisfiability", *Theoretical Computer Science*, pp. 49-64, N81(1), 1991.

[9] Littman M. L., Pitassi T., Impagliazzo R., On the Complexity of counting satisfying assignments, Unpublished manuscript.

[10] Greenhill, C., "The complexity of counting colourings and independent sets in sparse graphs and hypergraphs", *Computational Complexity*, 1999.

[11] Fürer, M., Prasad, S. K., *Algorithms for Counting 2-SAT Solutions and Coloring with Applications*, Electronic Colloqium on Comp. Complexity, Report No. 33, 2005.

[12] Dahllöf, V., Jonsson, P., "An algorithm for counting maximum weighted independent sets and its applications" ,*Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, pp. 292-298, 2002.

[13] Birbaum, E., Lozinskii, E.,"The Good Old Davis Putnam Procedure Helps Counting Models", *Jour. of Artificial Intelligence Research 10*, pp. 457477,1999.

[14] Roth, D., "On the hardness of approximate reasoning", *Artificial Intelligence 82*,pp 273-302,1996.

[15] Russ, B., *Randomized Algorithms: Approximation, Generation, and Counting*, Distingished dissertations Springer, 2001.

[16] Vadhan, S. P., "The complexity of Counting in Sparse, Regular, and Planar Graphs", *SIAM Journal on Computing*, pp. 398-427, V31, N2, 2001.

[17] Zhang, W., "Number of models and satisfiability of set of clauses", *Theoretical Computer Sciences*, pp. 277-288,N155(1), 1996.